A DISTRIBUTED NETWORK LOGGING TOPOLOGY

THESIS

Nicholas Eli Fritts, Second Lieutenant, USAF

AFIT/GCO/ENG/10-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCO/ENG/10-07

A DISTRIBUTED NETWORK LOGGING TOPOLOGY

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Nicholas Eli Fritts, BS

Second Lieutenant, USAF

March 2010

AFIT/GCO/ENG/10-07

A DISTRIBUTED NETWORK LOGGING TOPOLOGY

Nicholas Eli Fritts, BS

Second Lieutenant, USAF

Approved:

____//SIGNED//______________________	11 Mar 2010
Brett Borghetti, Lt Col, USAF (Chairman)	Date
____//SIGNED//______________________	9 Mar 2010
Dr. Robert F. Mills, (Member)	Date
____//SIGNED//______________________	9 Mar 2010
Dr. Michael R. Grimaila, (Member)	Date

AFIT/GCO/ENG/10-07

Abstract

Network logging is used to monitor computer systems for potential problems and threats by network administrators. Research has found that the more logging enabled, the more potential threats can be detected in the logs (Levoy, 2006). However, generally it is considered too costly to dedicate the manpower required to analyze the amount of logging data that it is possible to generate. Current research is working on different correlation and parsing techniques to help filter the data, but these methods function by having all of the data dumped in to a central repository. Central repositories are limited in the amount of data they are able to receive without losing some of the data (SolarWindows, 2009). In large networks, the data limit is a problem, and industry standard syslog protocols could potentially lose data without being aware of the loss, potentially handicapping network administrators in their ability to analyze network problems and discover security risks.

This research provides a scalable, accessible and fault-tolerant logging infrastructure that resolves the centralized server bottleneck and data loss problem while still maintaining a searchable and efficient storage system.

Acknowledgments

I would like to thank my advisor, Lt Col Brett Borghetti, for his help and patience in guiding me through to completion. I would also like to thank my friends and family that have helped support me during this process, it has been essential to my success.

Nicholas Eli Fritts

Table of Contents

List of Figures

List of Tables

A DISTRIBUTED NETWORK LOGGING TOPOLOGY

1. Introduction

Logging is a part of computer systems. Any administrator that has spent any time looking at logs is probably aware of two things: There are useful pieces of information in the logs, and it is normally not worth the time to try to find the useful information. The large quantities of data generated is one reason that log consolidation tools and log analyzers that help filter out some of the uninteresting information are currently significant research areas (Fu, Lou, Wang, & Li, 2009)(Halonen, Miettinen, & Hatonen, 2009).

Current research has shown that information on end user workstations can be used to help detect threats to the network (Levoy, 2006). Often, organizations set up a centralized server to capture log information from clients. However, a central log server solution could very quickly be overwhelmed by requiring all workstations report all logged system events to a central server. The bottleneck of communication with the central server must be addressed in systems that monitor the logs of an entire network in order for them to be effective.

This research develops a system where the work of maintaining external copies of logs is performed without any individual machine becoming a bottleneck for the log traffic. The structure of the system is scalable because the amount of work and traffic that a machine has to process does not depend on the network size.

Since the logs are stored on standard client machines which may sometimes be unavailable, the system is designed to ensure that log entries are stored in multiple locations. This reduces the chances of all of the copies of a log being simultaneously offline when they need to be accessed.

A method of searching the logs of the entire network is provided so that the data is easily accessible, and so that all of the responses take the same amount of work to be transmitted to the searching machine

The solution organizes machines in to small peer groups for log distribution and replication and arranges the peer groups in to a tree to provide an efficient method of searching the logs. It is designed so that all machines in a peer group are equivalent and all peer groups (except for the root) are equivalent

The thesis is organized into five chapters including this one. Chapter 2 is a literature review that covers related research in similar areas and provides historical information on the subject. Next, Chapter 3 provides details for the log protocol and the experiments to test it. Chapter 4 presents the results of the experiments and discusses how the system performed. Chapter 5 summarizes the results and discusses what areas could be expanded in future work.

2. Literature Review

2.1. Overview

This chapter provides information on computer security and the threats against that security. Section 2.2 will focus on computer and information security and discuss some of the tradeoffs associated with making a system more secure. It will then discuss some of the threats against information systems and highlight the insider threat. Section 2.3 will provide a definition of insider threat and then describes some of the history behind it and various detection methods. Section 2.4 discusses detection methods and specifically the usefulness of auditing and logging. Section 2.5 discusses various peer-to-peer protocols that could be applied to network logging.

2.2. Security

Computer security has been a focus since the first worm appeared and shocked the internet in 1988 (Spafford, 1989). Since then many advances have been made in the realm of computer security, and computer security can still be condensed in to three basic goals.

2.2.1. Goals of Computer Security

Computer security consist of three goals often referred to as the “CIA” model. The anagram CIA comes from the first letter of the three goals: Confidentiality, Integrity and Availability (Federal Information Security Management Act, 2002). The Federal Information Security Management Act of 2002 (FISMA) defined the goals for use by government agencies. FISMA defines Confidentiality as “preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information.” Integrity is defined as “guarding against improper information modification or destruction, and includes ensuring information nonrepudiation and authenticity.” Availability is defined as “ensuring timely and reliable access to and use of information (Federal Information Security Management Act, 2002).”

Though those three goals were codified by Congress in 2002, they have existed for much longer. In fact the National Security Telecommunications and Information Systems Security Committee released an advisory memorandum on the threat from insiders to government networks in 1999 that mentioned using the CIA model as a baseline for developing ways to judge computer security methods (Hayden, 1999). The CIA model is based on the idea of authorization. Under the model there is information, and there are people accessing the information. The people accessing the information are authorized to do so, otherwise a breach in security has occurred (Denning, 1999).

2.2.1.1. Confidentiality

Confidentiality as defined above is simplified to fundamentally mean the information is available on a "need-to-know" basis. For example, when a customer calls a phone company about their service, the phone company generally asks the customer to prove who they are by answering personal questions. This proves to the phone company that the person is the owner of the account and has access to the confidential account information. Anytime there is a situation dealing with financial or medical matters basic measures are taken to ensure the confidentiality of data. Breaches of confidentiality have been happening to approximately 9% of the respondents of an annual computer crime survey for the last five years (Richardson, 2008). When organizations have laptops with sensitive data stolen, or backups taken from someone's car; these are examples of a physical breach of confidentiality. With the exception of having physical devices stolen, in a computer system confidentiality is often enforced via permissions on files and login credentials which will either allow or deny access to sensitive information based on who a user told the system that they are.

2.2.1.2. Integrity

Integrity deals with making sure that the data is accurate and not modified without authorization. Many areas depend on integrity and one primary example is a police investigation. All evidence in a police investigation must be forensically sound. That is that the integrity of the evidence must not be in question. If the evidence is a computer hard drive, investigators must be able to show that they did not modify the files on the drive to get the evidence. When threats against a network are realized, often the threat will attempt to modify system logs or files to hide the fact that they have been there (Hayden, 1999). If a user can modify the system to remove all traces of their presence on the system then the integrity of the system and potentially the data it contains has been compromised.

2.2.1.3. Availability

Availability refers to the ability of people who are authorized to access and use information in a timely manner. There are many threats against availability and some of them have no mitigation strategies. If an earthquake or other natural disaster destroys the power infrastructure to an information system, then even if the data is still intact, it is not available to the people who need it and in a general sense, the security of the system is compromised. Another common threat to availability is a Denial of Service (DOS) attack which aims to consume or disable resources on a server in order to prevent normal access of services.

2.2.2. Impact on Usability

It is generally accepted that there is a tradeoff between computer security and usability. Security researcher Dorothy Denning wrote “the only way to make a computer system secure is to pull the plug (Denning, 1999).” She recognizes that the goal of information is not to deny access, but to instead permit authorized access, however there is a difference between writing a policy to define who has access, and then implementing and enforcing such a policy on a computer system.

2.2.3. Types of Threats

There are three main threats to a network: Natural, External, and Internal. There is little that can be done to protect an information system from natural threats such as storms, earthquakes, or other natural disasters aside from housing backup systems at contingency locations. Therefore, for the purpose of this research, natural threats to information systems will not be focused on. The next threat, external, is essentially focused on hackers, competitors, or foreign agents that are working from outside a network with only publically available information. These threats are up against an organization’s largest defenses, because most organizations spend most of the information security budget to protect from external sources (Mills, Peterson, & Grimaila, 2009). This research assumes there are other measures in place to protect from and defend against external threats, however if an external threat gains access to the network it is possible that they could then be considered an internal threat. There are additional external defense methods that could detect them, but once they have penetrated the organizations defenses, they may begin to act as an insider would. Internal threats are generally considered to be the greatest risk to information systems (Mills et al., 2009). Internal threats come from “insiders” who are generally current or former employees or business partners who have detailed information or have/had authorization to access information not generally available to the public (Greitzer et al., 2008). Internal compromises are normally less frequent that external compromises, but due to the elevated level of trust, often have a significantly higher damage cost (Mills et al., 2009).

2.3. Insider Threat

Frank Greitzer from Pacific Northwest National Laboratory states that an insider "is an individual currently or at one time authorized to access an organization's information system, data, or network" (Greitzer et al., 2008). While the National Security Telecommunications and Information Systems Security Committee (NSTISSC) said in their Advisory Memorandum 1-99 that "Insiders can be employees, contractors, service providers, or anyone with legitimate access to a system." There are many variations in the precise definition, but the major points: authorized access, and possession of knowledge not publically available are generally agreed on. It is worth noting that insider threats are not always intentional. Users who may just have a goal of making it easier to do their job may do such things as installing shareware, disabling virus protection, or using unapproved storage devices; and those actions may provide an external threat with access to the network (Hayden, 1999).

2.3.1 Historical Insider Threat Information

The insider threat problem has been around for a long time. Mills acknowledges that insider incidents are often unknown or go unreported, however there are published events of insider issues relating to government networks dating back to

1988 when a Libyan intelligence agent was able to access information on government employee car pool information through his wife's employer (Hayden, 1999). Common places where insiders misuse information or job resources are hospitals and government databases where sometimes the insider simply wanted to see what treatment a celebrity was receiving, or how much their family was paying in taxes (Hayden, 1999). It is clear that the insider threat has been around for a while, and thanks to recent surveys, organizations are beginning to notice problems with insiders and do something about them (Richardson, 2008).

2.3.2 Insider Identification

Preventing insiders from causing problems would be ideal, but since preventing everyone who works for an organization from ever doing anything wrong is unrealistic; the focus goes to identifying when the behavior has changed as quickly as possible, and helping the organization to understand the damage and respond. In the historical cases presented by the NSTISSC, most of them were identified when looking back at audit logs (Hayden, 1999). Levoy developed a methodology to tune the logs of a stand-alone Windows XP workstation in order to identify a set of cases he determined would simulate insider actions while minimizing the amount of logging that an administrator would have to look through (Levoy, 2006). It is generally accepted that to identify insiders, "observables" must be documented, and if possible correlated (Mills et al., 2009)(Hayden, 1999).

Levoy's work was one of the first of its kind in aiming to optimize a computer log for detecting insider threats. His work was groundbreaking, but limited in that it only analyzes a single source of information, and requires an administrator to predetermine what threats their network faces.

2.4. Auditing and Logging

According to a SANS Institute survey on logs, most organizations that collect audit logs use them for detecting and analyzing security and performance incidents, and almost half the organizations use their logs for some kind of standards compliance reporting (Gordon, Loeb, Lucyshyn, & Richardson, 2006). Logs can provide the information that is used to assess and test a network, identify areas that need repairs, or identify and stop an intrusion (Shenk, 2008). Logs also can track individual accountability or help construct a timeline when an event happens on a network (Ma & Tsudik, 2009).

2.4.1. Current Use of Logging

Logs were originally developed for programmers and system administrators to debug systems and applications (Peisert, Bishop, & Marzullo, 2008). Today most systems have at least three parts: the generator, the log subsystem and the archival system. The first is where the event is generated. This can be a program running on the system, or debug code in the kernel of the operating system. This is where the event actually occurred and makes a local call on the system to send the information to the logging subsystem. The logging subsystem handles the logs and sends them to the archival subsystem. In networked systems, the archival system might be a centralized log server and so the log subsystem must transmit the event across the network infrastructure or queue the events if the network is non functional. The archival system is designed to receive messages and write them to disk. Ideally this is done in a method that is forensically sound, however most current methods do not provide the kind of forensic integrity required of legal evidence (Monteiro & Erbacher, 2008).

2.4.2. Logging Best Practices

For logs to be less confusing and of the greatest benefit to an organization, they should have three main characteristics: A synchronized time stamp, a sufficient level of detail to identify what event occured, and sufficient archival logs to get more information if required (GadAllah, 2004). The time stamp is most important when networked systems are involved because most systems put the time stamp on the logs when the event is created, not when it is archived. If events are being correlated between different machines and the times on the machines are different, it can be difficult to link events between the machines. Sufficient detail is required, because knowing that an event happened is not always enough. For example, if a user “Joe” logs in to a machine and a “login” event is recorded, the event is useless for later analysis unless we know WHO logged on and WHERE they logged on. The archival information is important because in the example above if the user logged on to a system and performed an action that he should not, it might be useful to see that fifteen minutes previously, Joe logged in from Texas, and the logon where policy was violated logged in from China (GadAllah, 2004).

Logs are primarily used for security and performance analysis, but depending on the industry, policy compliance can drive the logging policy. Half of SANS's survey respondents did say that the retention policy was driven by standards compliance (Shenk, 2008). One problem with logs is that they are not usually secure. Ideally if a log server (or any machine with logs) is compromised, the attacker would be unable to modify or read logs that were created before the machine was compromised. However, Schneier points out that "no security measure can protect the audit log entries written after an attacker has gained control of [the system]" (Schneier & Kelsey, 1999).

2.5 Peer-to-Peer Networks

Peer-to-peer networks allow the distribution of network functions and storage so that in effect the storage and function of the network is in the network cloud and no longer dependent on individual machines. There are many different peer-to-peer protocols in existence, but some popular ones are GNUTella, KaZaA and BitTorrent (Karrels, Peterson, & Mullins, 2009). Peer-to-peer systems are a method of storing, retrieving or streaming data in a distributed way, and could potentially be applied to logs.

The main difference among the current generation of peer-to-peer technologies is the structure of the network. Initial networks used a method of broadcasting queries to all peers that ended up scaling poorly (Karrels et al., 2009). Newer protocols such as KaZaA and GNUTella use a topology where an election process is used to establish super nodes in the network and those supernodes provide a second level of organization that helps keep searching and content location more efficient (Frankel & Pepper, 2008; Leibowitz, Ripeanu, & Wierzbicki, 2003).

BitTorrent is setup for cooperative distribution of large files. It works by splitting a file in to pieces and then allowing anyone who has a piece of a file to send it to the machine trying to download it. The machines get in touch with each other via a "tracker" which is a machine that maintains a list who is currently downloading or uploading the file. When a file is being downloaded, the tracker will give the downloader a partial list of the machines it should contact to download pieces (Cohen, 2003).

One complexity of optimizing network logging is that content flow traditionally runs in the opposite direction from a normal client-server architecture. In a normal architecture, the server has content and it is generally being pulled to the clients. In a traditional logging task, clients are creating content and sending it to the server. When looking at distributing these log files, neither peer-to-peer nor content delivery networks are designed to handle this reverse flow.

Peer-to-peer networks are so prevalent today because of their ability to scale and function in an extremely large network with an unstable population with minimal central architecture (Androutsellis-Theotokis & Spinellis, 2004). One of the biggest costs in peer-to-peer networks is the routing tables, and the tradeoff between table size and how many messages must be sent to find files on the network (Androutsellis-Theotokis &

Spinellis, 2004). At their simplest form, peer-to-peer networks and applications are designed to utilize the resources from the network that would otherwise be unused (Androutsellis-Theotokis & Spinellis, 2004).

Content Delivery Networks (CDN) are designed to be a set of distributed servers that dynamically allocate users to a closer, and preferably less utilized server than the primary server hosting the content (Pallis & Vakali, 2006). Recent approaches have begun to create a hybrid between CDN and peer to peer networks. They hope to take some of the advantages in peer-to-peer networks such as the scalability and fault-tolerance and overcome the weaknesses such as the performance being dependent on number of peers to create a more efficient system (Jiang, Li, Li, & Bai, 2008). One of the methods used to redirect users to a closer server involves DNS redirection, which allows for a client to be redirected from the beginning of their communication with a server, and is often used as a form of load balancing (Krishnamurthy, Wills, & Zhang, 2001). One use of a CDN type network is to use multiple central tracker's for the BitTorrent protocol so that if trackers fail, the service will still be usable (Li, 2008).

A hybrid based on the CDN and P2P technology should make a flexible and scalable network, but in order to apply the technology to network logging, the fact that content flows from client to server needs to be addressed.

3. Methodology

This chapter discusses the methodology that will be used to test distributing network logs across all machines in the network to add scalability to the logging infrastructure. Section 3.1 discusses some of the challenges with distributing network logs and the desired characteristics of a solution. Section 3.2 discusses the details of the proposed solution and the various operations that machines must support. Section 3.3 discusses how the system will be simulated and the various scenarios that will be tested. Section 3.4 discusses what data has to be collected from the simulations and how the system will be evaluated.

3.1. Problem Description

In order to ensure that every log entry is stored and replicated in a place other than the machine that generates it, some level of organization is required to prevent the logging infrastructures from becoming a fully meshed topology where every machine is transmitting to every other machine. A fully meshed topology would result in every machine being unable to handle the log entries it receives. One of the challenges is that every machine is generating a continuous stream of unique entries to be stored. This chapter presents a solution to distributing network logs using peer-to-peer topologies. Using peer-to-peer techniques is ideal because they provide scalability and allow machines that are generating logs to share the workload for storing and distributing the logs.

3.1.2. Desired Characteristics of a Solution

An ideal solution to the problem would have several desirable qualities. First, it should be designed so that it does not require significant processing power from individual machines. Since the solution would be running on end user equipment and not just dedicated servers, it must not prevent the machines from being used for their primary purpose. Second, the log entries must be available when needed. One of the problems with running logging infrastructure on a system that is not dedicated to that purpose is that it may not be online when you need information that is stored on that node. An ideal system will address nodes being unavailable by ensuring that the probability of data being unavailable is low. Third, the amount of work that a machine in the network does should be approximately equal to any other machine on the network, and should be independent of the total number of machines on the network. Fourth, an ideal system will minimize the time and number of machines that must be communicated with to search the system. If a proposed system requires the machines making a query to probe every machine on the network, then searching the logs will not be efficient and even though the logs may be distributed efficiently and redundantly, accessing the logs will not be efficient enough to allow retrieving the logs.

3.2. Solution: A Tree of Peer Groups

The solution to the problem is a tree structure that is made up of peer groups of machines. Each peer group is a set number of machines that work to ensure that every machine has a record of all the log entries from every machines in the peer group. The peer groups are organized in to a tree so that there is an efficient method to perform network-wide searches for log entries. An example of this topology is shown in Figure 1.

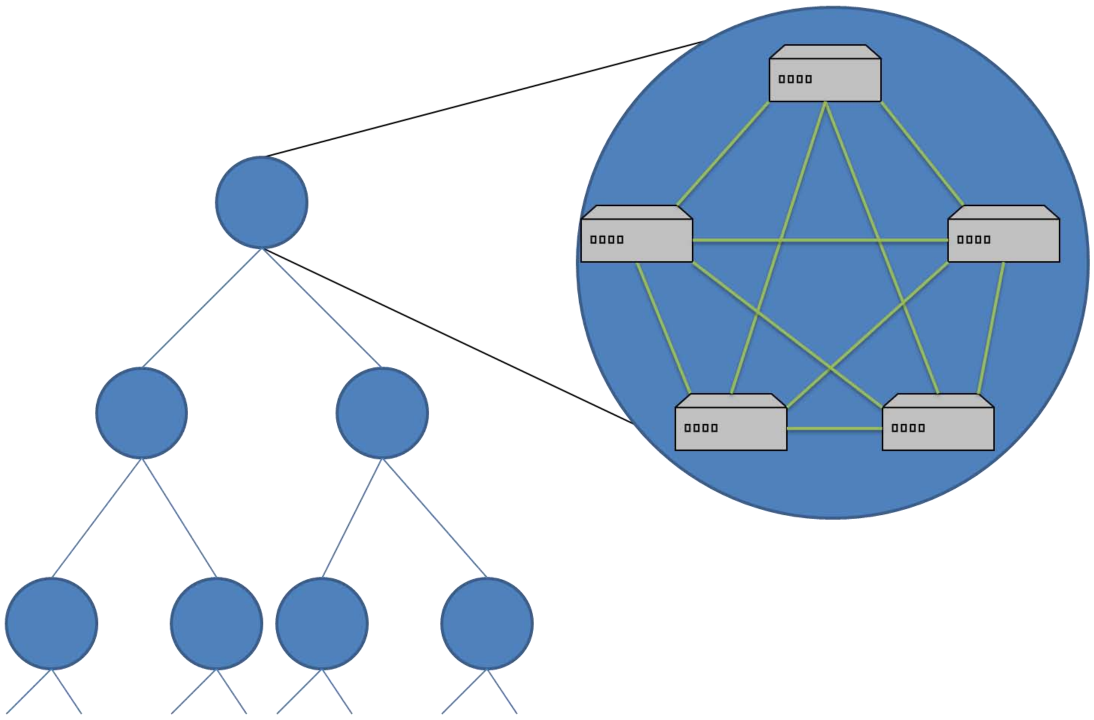

Figure 1:Topology with peer groups represented by circles and an expanded view of peer group communication in the large circle.

The solution involves organizing all of the machines generating logs into peer groups of a set size and having the organization of the peer groups be represented by a tree that allows searches. The number of “backups” of log entries is equal to the size of the peer group (which is configurable). If there are four machines in a peer group, then there will always be at least four copies of every log entry made by a machine in that peer group.

In order for the topology to work, there are several operations that must be defined and implemented. These operations are divided in to two different categories: Tree/Peer Group operations and Peer Group/Machine operations. The Tree/Peer Group operations operate at a level above the individual machine level, and any time communication with a peer group must occur, one machine in that peer group is selected in a manner to ensure that each machine in the peer group is responsible for an equal portion of the different messages. One method to balance which machine is doing the communication would be to use a uniform distribution random number generator to select the leader each time. Any time a peer group is talked to, one machine is going to handle the communication, but after the message is passed, most events will require notifying the rest of the peer group so that any state information for the peer group is kept in sync between the peer group members.

The operations that are supported are:

- Finding the root
- Adding machines to the network
- Adding peer groups to the network
- Removing machines from the network
- Removing peer groups from the network
- Distributing the logs
- Searching the log

These operations are described in detail in the following sections.

In order to specify the proposed solution, several variables are defined to represent different parameters of the protocol and the network. These variables are detailed in Table 1.

Table 1 Protocol Variables

N	Total Number of Machines on the Network	Integer
n	Minimum Peer Group Size	Integer
b	Peer Group Buffer Percentage	Float
R	Rate of Messages Generated by Machines	Float
$RebootTimeout$	Time a Machine can be Offline and still Remain in a Peer Group	Integer
$MemberTimeout$	Time a Machine can not respond before starting the $RebootTimeout$	Integer
n_{min}	Minimum Peer Group Size (n)	Integer
n_{max}	Maximum Peer Group Size $(n + bn)$	Integer
PG_N	Number of Peer Groups in the Network (measured)	Integer

3.2.1.1. Finding the Root

In order for the machines to join the network and to support the operations below, machines need to be able to find the root peer group. Since machines in the root peer group end up doing more work than other peer groups, machines are cycled out as explained in Algorithm 2. To support the node cycling at the root, there are two different methods presented that would ensure a machine in the root can always be found. The first is to have a dedicated machine that stays in the root and handles all root requests. This would be effective, but implies a single point of failure for tree based operations that need to find the root. A more robust solution is to use a “fast-flux” type Domain Name System (DNS) configuration to ensure that an entry for the tree root resolves to the machines that are currently in the root peer group and does not allow caching of the result

(Holz, Gorecki, Rieck, & Freiling, 2008). As machines leave the root peer group, they would be removed from the DNS revolver rotation. This allows any machine on the network to find machines in the root peer group with a DNS lookup.

3.2.1.2. Adding Machines

For a machine to be added to the network, the joiner will perform the operations in Algorithm 1:

Algorithm 1 ***Join_Network Pseudocode***

```
1.  Procedure Join_Network
2.      Begin
3.      Joining machine ($J$) looks up a member of the root peer group ($R_m$).
4.      $J$ sends a request to join the logging network to $R_m$.
5.      $R_m$ sends the root peer group information to $J$.
    //Information includes the member list, growth flag and children information.
6.      $R_m$ tells the other root members that $J$ is a new member.
7.      Root PG performs a log redistribution (See Algorithm 8)
8.      If the root peer group has $2n$ or greater members Then
9.              Perform Algorithm 2
10.     End If
11.     End Procedure Join_Network
```

Algorithm 1 grows the root peer group and allows for members to join the logging network quickly. The root peer group is the only peer group that grows in size as time progresses. Once the root peer group has grown to $\geq 2n$ members, it creates a new peer group and inserts it in to the tree according to the steps in Algorithm 2:

Algorithm 2 *Create_Peer_Group Pseudocode*

```
1. Procedure Create_Peer_Group
2.     Begin
3.     $R_m$ from Algorithm 1 adds a member to the root peer group
4.     The root peer group is now $\geq 2n$ members.
5.     $R_m$ sends a message to all root members identifying the $n$ oldest members.
6.     The $n$ oldest members are moved to a new peer group and passed down the tree
           according to Algorithm 3.
7.     The remaining members are identified as the root peer group.
8.     End Procedure Create_Peer_Group
```

In order to keep the tree balanced as peer groups are removed and added, each peer group has two children and a flag to know if the last time it performed an insertion it sent the peer group to the left or right. For every peer group that is passed down one side of a given peer group, the next peer group passed down to that same peer group will go down the opposite side. This ensures that the difference between number of children to the left or right of a given node is at most one[1]. When an insertion is done, Algorithm 3 is called with two parameters (the new peer group to be added to the tree, and the peer group identifier of the left of right child of the root (whichever side is being grown).

[1] If the flag for which side to grow is tracked as a binary representation of an integer counting the number of nodes added to the tree, the LSB serves as the flag, with odd numbers on one side and even on the other. If the number is even, the two sides have the same number of children, if odd, they differ by 1. When the number is a power of 2 minus 1, the tree is a complete tree.

Algorithm 3 *Insert_Peer_Group Pseudocode* (PG_{New}, PG_{Cur})

```
1.  //This procedure would be called initially at the root as described in Algorithm 2
2.  Procedure Insert_Peer_Group
3.      Begin
4.      If $PG_{Cur}$ has < 2 children Then
5.          $PG_{Cur}$ claims parenthood of $PG_{New}$ and toggles its growth flag.
6.      Else
7.          $PG_{Cur}$ Reads it's growth flag to determine which child to grow ($PG_C$).
8.          $PG_{Cur}$ Toggles it's growth flag.
9.      End If
10.     Insert_Peer_Group($PG_{New}$, $PG_C$) is called.
11.     End Procedure Insert_Peer_Group
```

3.2.1.3. Removing Machines

There are two different cases that must be handled for removing a machine from a peer group. The two cases are that 1) the machine leaves gracefully and notifies its peer group that it is going offline, and 2) the machine goes offline without notification and stops responding to requests and log entries from the peer group. For both cases, the only difference is at what point the machine is considered lost and the peer group analyzes if it has enough members to continue its existence. In case 1 if a machine signals that it is going offline, a timer is started to give it time to reboot for maintenance and updating without having to rejoin the tree. This will be called the $RebootTimeout$ and should be configured based on desired network behavior. Once the $RebootTimeout$ has elapsed without the machine coming back online, the machine is considered lost and when it comes back, it will have to rejoin the tree. Case 2 occurs when a machine fails to respond to other machines in the peer group within a certain timeframe called the

$MemberTimeout$. The $MemberTimeout$ should be configured based on a specific network's expectations of regular maintenance and system reboots. After the member timeout expires, the machine is considered offline and the peer group will start a timer using the $RebootTimeout$ to determine when the machine is considered lost.

When a machine has been offline for the length of the $RebootTimeout$, that machine's peer group performs steps outlined in Algorithm 4.

Algorithm 4 ***Peer_Group_Member_Timeout Pseudocode***

```
1. Procedure PG_Member_Timeout
2.     Begin
3.     If Current Peer Group has < $n$ members Then
4.         Current Peer Group notifies the root that it needs to be removed.
5.     Else
6.         Do Nothing.
7.     End If
8.     End Procedure PG__Member_Timeout
```

3.2.1.4. Removing Peer Groups

Peer group removal also has two potential cases. The first case is that the one of the peer group members signals the root to remove it since it does not have enough members, and the second is that the entire peer group goes offline simultaneously. Removing peer groups must be done carefully in order to ensure that the tree remains balanced. To ensure the tree remains balanced for the first case, the root performs a traversal of the tree to find the peer group that was last added to the tree. The traversal is done by checking the $GrowthFlag$ and traversing the opposite child from the one indicated until a leaf node is found. Every time the $GrowthFlag$ is checked, it is also toggled so that the next peer group to be added will end up in the same position as the leaf node that is found. The steps for finding the leaf peer group are defined in Algorithm 5.

Algorithm 5 ***Find_Leaf_PG Pseudocode***

```
1.  Procedure Find_Leaf_PG
2.      Begin
3.      If Root Peer Group has < 2 children Then
4.          Root Peer Group's child is the leaf.
5.      Else
6.          Toggle Growth Flag.
7.          Set $PG_S$ to child indicated by the Growth Flag
8.          While $PG_S$ is not a leaf do
9.              Send request to $PG_S$ for a leaf node.
10.             If $PG_S$ has no children Then
11.                 $PG_S$ sends response that it is a leaf.
12.             Else
13.                 $PG_S$ toggles it's own growth flag.
14.                 $PG_S$ sends child indicated by growth flag to the Root.
15.             End If
16.         End While
17.     End If
18.     End Procedure Find_Leaf_PG
```

A peer group swap between the leaf and the dying node is then performed as outlined in Algorithm 6.

Algorithm 6 ***Swap_Peer_Group Pseudocode***

```
1.  Procedure Swap_Peer_Group
2.      Begin
3.      One machine is identified in each peer group as the coordinator for the swap.
4.      The two coordinators exchange parent, children, and growth flag information.
5.      The coordinators sync that information to the rest of their respective peer groups
6.      The coordinators signal to each other that the distribution is complete and
            confirmed.
7.      The coordinators signal their peer groups to use the new information
8.      The coordinators signal each other that the transfer is complete.
9.      The coordinators signal the root that the swap is complete.
10.     End Procedure Swap_Peer_Group
```

Once the peer group swap has completed, the members of the dying peer group (which is now the leaf of the tree) each rejoin the network according to the add machine algorithm placing them in the root of the tree. Removals must be done atomically with respect to creating new peer groups so that a new peer group will not be sent down the tree while a leaf is being found to facilitate a removal. To ensure the atomic nature of removing peer groups, the root will not make any new peer groups while finding a leaf peer group and before receiving confirmation that the peer group swap has been completed. It will also queue any other peer groups that need to be removed from the tree until the operation completes.

The case of all machines in a peer group going offline simultaneously is more complex because it segments the tree. To maintain the integrity of the tree, a heartbeat type signal is established between the peer groups. Every few seconds, each machine in the peer group checks the status of one machine in each of the parent and children peer groups. A tree segmentation is noticed when either the parent of the dead peer group check their children or the children of the dead node check their parent. When the parent of the dead node notices that a peer group has gone offline, it notifies the root node who acknowledges it but waits for the children of the dead node (which are now the roots of segmented portions of the tree) to contact the root. The children upon noticing that their parent is no longer responding lookup the root peer group and signal that they are the root for a segment of the tree. Once the root peer group is notified that a segment of the tree has been located the steps shown in Algorithm 7 are performed.

Algorithm 7 ***Repair_Tree Pseudocode***

```
1.  Procedure Repair_Tree
2.      Begin
3.      The root peer group halts the growth of the current children.
4.      Root peer group identifies current children and orphans that have contacted it as
              temporary children
5.      The growth flag for the root peer group is reset to the initial state.
    // At this state, the root peer group has temporary children, but no left or right child
6.      While temporary children exist do
7.          Find leaf node on one of the temporary children (Algorithm 5)
8.          Each found leaf is removed from the temporary branch
9.          Each found leaf is inserted to the tree according to Algorithm 3 using
                  the roots new left and right children.
10.     End While
11.     End Procedure Repair_Tree
```

Algorithm 7 removes all peer groups from the branches of the tree that were disconnected and places them in new balanced branches off of the root peer group without having to perform any log redistribution.

3.2.1.5. Log Distribution

Log distribution is performed at the peer group/machine level and does not involve machines outside of the peer group. Under normal operations, as a log entry is generated on a machine, it will be sent to the members of the peer group as if they were log servers themselves. They then record the message.

In the case of an interruption of service for a machine where it is considered offline but comes back within the window of time before it is forced to rejoin the network, the log entries that the machine missed are condensed to one “file” and then transferred in a BitTorrent type manner to the machine that went offline while the machine that went offline does the same for any log entries that it needs to distribute. Logs are condensed into easily distributable files that cover a specific configurable time interval. When a system needs to be caught up or redistribution is performed, these consolidated log files are what is distributed and then the logs that have not been condensed are sent separately.

The last piece of distribution is performed when a machine enters a new peer group (which includes joining the root node). When log redistribution is performed, Algorithm 8 is used to synchronize each machine in the peer group.

Algorithm 8 ***Distribute_All_Logs Pseudocode*** (M)

```
1.  // M is the machine to distribute logs from.
2.  Procedure Distribute_Log_Event
3.      Begin
4.      Create a list ($L_m$) of every machine that currently has entries on $M$.
5.      For each machine($M_x$) in $L_m$
6.          If $M_x$ is online and not in the current peer group Then
7.              Delete entries for $M_x$
8.          End If
9.      End For
10.     Create a file to be distributed that has all remaining log entries.
11.     Distribute that file to all Peers.
12.     End Procedure Distribute_Log_Event
```

NOTE: File distribution should be done in an efficient peer to peer method such as BitTorrent that allows for asynchronous distribution of the file.

3.2.1.6. Log Searches

Searching the logs is performed utilizing the tree structure the peer groups are organized into. The search can be performed from any machine on the network, and the searching machine is treated as if it was external to the tree. While the query is traveling down the tree, the results are sent directly to the querying node without traversing the tree. The searching machine can end up receiving results from more than one peer group and more than one machine, depending on the query that is sent out. The query progresses along the steps in Algorithm 9.

Algorithm 9 ***Search_Logs_Peer_Group Pseudocode*** (SM, Query, $PG_{Current}$)

```
1. Procedure Search_Logs_Peer_Group
   //Searching Machine (SM) initiates Search_Logs_Peer_Group(SM,Query, $PG_{Root}$)
2.     Begin
3.     $PG_{Current}$ determines the query result using Algorithm 10.
4.     The $PG_{Current}$ sends SM the answer to that query and how many children it
           sent the query to.
5.     $PG_{Current}$ Initiates Search_Logs_Peer_Group(SM, Query, $PG_{Child}$) on all children
6.     End Procedure Search_Logs_Peer_Group
```

NOTE: Peer groups determine the answer the query according to Algorithm 10.

Algorithm 10 ***Search_Logs_Machine Pseudocode***

```
1.  Procedure Search_Logs_Machine
2.      A machine ($M_1$) in the peer group receives a query
3.      Begin
4.      $M_1$ sends the query to all peers
5.      $M_1$ queries its own logs and hashes the result
    //  All other peers query their respective logs and hash the result
    //  All other peers send the hash of the query result back to $M_1$.
6.      If all hashes match Then
7.              $M_1$ sends the results of its query to $SM$ and flag that all members agree.
8.      Else
9.              $M_1$ requests full results from all other peers
10.             $M_1$ receives all other results
11.             $M_1$ sends all results with which machine generated them to SM.
12.     End If
13.     End Procedure Search_Logs_Machine
```

NOTES: Hashes are used to minimize the amount of information transferred when logs match. All non-matching results are sent to SM to help minimize secondary searches due to mismatched logs.

Algorithm 9 results in the SM receiving either a no results match and how many children a peer group has or a set of results and how many children the peer group passed the query to for each peer group. Telling the SM how many children the search has been passed to allows the SM to know when the search is complete and give an estimation of how many results are pending. The results are not sent up the tree because peer groups do not change places in the tree very often. If log entries were sent up the tree, the amount of traffic that each peer group would have to transmit would grow exponentially based on how high it is in the tree. This is because each node is going to have $2(2^x - 1)$ children below it where x is the number of full levels below the current node.

3.2.2. Refinement

Since peer groups do not grow in size, any time a member goes offline, the rest of the peer group must be moved. Since this causes extra work and movement in the tree, the topology will be modified to include a number of buffer machines that will attempt to minimize the impact of machines leaving a peer group.

3.2.2.1. Buffered Peer Groups

The original topology creates peer groups at the minimum size (n). An effect of building peer groups at the minimum size is that every time a machine leaves a peer group, that peer group is going to remove itself from the network which will result in every other machine in that peer group performing a peer group change which is expected to be the most expensive operation for a peer group to perform. In order to minimize the number of peer groups that get removed, a number of buffer machines is added to each peer group. The optimization is done by instead of creating peer groups of size n, only creating them once there are $\lceil n + bn \rceil$ machines to put in a peer group where b is a percentage of extra machines to provide a peer group as a buffer.

3.3. Simulation

In order to compare the non-buffered and buffered approaches against the standard central server topology, a simulation in MATLAB is used. Various scenarios are outlined in this section describing how the system was tested.

3.3.1. Tree/PG Simulations

The Tree/Peer Group simulations focus on topology formation and maintenance where the peer groups are mostly considered to be atomic. The network can grow and shrink by adding or removing individual machines, but most analysis is done on the organization of the peer groups and the effect this has on the tree. The exception is the peer group changes which are tracked for each machine.

3.3.1.1. Random join/rejoin Event Scripts

The simulations here are designed to be repeatable by creating pseudo-random event scripts and recording them so that different configurations, topologies and types of operations can be tested with the same random series of events. The script is a matrix that has a column for each machine on the network and a row for each time-step. The matrices are created by assigning each cell a random number from a uniform distribution that represents the probability of the machine changing states. The scripts will be populated and show when each machine joins and leaves the network to allow multiple machines to leave or join the network at the same time. The simulation prioritizes actions such that from highest to lowest probability the actions are: machines leaving, peer group changes, and machines joining. All actions are resolved before the simulation goes to the next step.

The parameters that are kept the same for all of the scenarios are listed in Table 2.

Table 2 Simulation parameters that are the same for all scenarios.

N	10000
T	3000
n	4,6,8,10
b	0,.25,.5

The n and b parameters are combined to perform a full factorial experiment so that every chosen n parameter is run with every chosen b parameter.

3.3.1.2. Event Script Parameters

To determine what a network would look like in the morning when most people are arriving at work, data was collected with the help of the communications squadron from the Air Force Institute of Technology from their network for analysis. The data used is from January 19, 2010 and contained information on what time machines were logged in to and how long they had been on. Due to AFIT scheduling, the morning was defined as the first login of the day until 1055 hours. In order to simplify analysis, this time was divided in to 5-minute blocks, and it was observed that an average of 0.6657% of machines all machines were logged in to during each 5 minute block. This was used to set α to 0.006657. It was also observed that of all the machines that were logged in to, 17% of them were rebooted. The percentage of machines that rebooted was used to configure δ to be 0.17α or 0.00113543.

3.4. Evaluation Techniques

The evaluation of the approach will focus on measuring the factors that impact search complexity, log redundancy, event throughput and network usage and the fairness of the system and to compare the baseline of a central server against the new distributed topologies.

3.4.1. Search Complexity

Search complexity looks at the number of peer groups in the network and how many responses an administrator trying to query the logs will have to receive before having all of the results. This is determined by the number of peer groups and how many machines are online/offline at the time of the query. Searches traverse the entire tree and require every peer group to send a response to the searching machine. In order to measure search complexity, the average number of peer groups in the network will be measured. The worst case for any topology is having every peer group at the minimum size which will result in the searching machine receiving $\frac{N}{n_{min}}$ responses. The best case for a network is when all peer groups are at the max size which results in $\frac{N}{n_{max}}$ responses.

3.4.2. Redundancy

Redundancy for this topology is a measurement of how many copies of a log entry exist. This is specifically measured by recording the size of all the peer groups which is how many copies of each entry exist and how many machines would have to go offline simultaneously to cause the loss of the log entries. This will be compared to a central server where logs primarily exist in one location. The best and worst case can be easily determined by looking at n_{min} and n_{max}.

3.4.3. Event Throughput and Network Usage

This measures the maximum amount of traffic that any machine in the network will be expected to handle as a function of peer group size. This will be compared to the throughput on the central server that must handle all of the logging traffic. The proposed topology has two effects: 1) At the machine level, total traffic from logging is expected to grow linearly with a function of peer group size, and 2)The actual traffic that is going on inside the peer group will increase quadratically with peer group size. Logging traffic is calculated with respect to R, and is one event being passed from a machine to the peer group. Depending on the criticality of the logs on the network, different exchange methods could be implemented: unreliable UDP transfer, reliable TCP transfer, or potentially an encrypted SSL transfer. Each method would have different levels of traffic, but would result in a constant increase in traffic that has R as the dominant factor. The minimum traffic a machine will see exchanging one log event is $2(n_{min} - 1)$ and the maximum is $2(n_{max} - 1)$. The total logging traffic in a peer group for one log event is $n^2 - n$ which gives us a peer group minimum of $n_{min}^2 - n_{min}$ and a maximum of $n_{max}^2 - n_{max}$. The entire network logging traffic to be determined to be $PG_N\left(n_{avg}^2 - n_{avg}\right)$. PG_N and n_{avg} are measured during the simulation, but the minimum logging traffic on the network can be calculated by substituting variables and using the equation $N(n_{min} - 1)$. The worst case logging traffic is determined by $N(n_{max} - 1)$.

3.4.4. Fairness

Fairness is evaluated to ensure that regardless of a machine's peer group membership and the peer group's location in the tree, it is not required to do significantly more work than the rest of the machines on the network. One purpose of this is making sure that peer group changes happen approximately equally to all machines in the network. A second purpose of this is ensuring that searches do not require peer groups to do work for their children. The second measure has been minimized by the design of the topology, and the first is measured by tracking how many times machines have to change peer groups. Changing peer groups is the primary measurement because log redistribution is the most expensive operation and is primarily performed when peer groups are created, or a peer group is given new members (adding members only happens to the root). This is why machines that have been in the root node the longest are moved to the newly created peer groups first. To evaluate fairness, the minimum number of peer group changes is subtracted from the maximum number of peer group changes. This shows the difference between the machine that experienced the best treatment (low number of peer group changes) and the worst treatment (the highest number of peer group changes). It is expected that machines will experience an average of at least two peer group changes every time they join the network, because when a machine joins, it will “change” peer groups to the root, and once enough machines have joined, change peer groups to a new peer group that is sent down the tree.

3.5. Summary

The next chapter analyzes the results of simulation runs and compares the networks to theoretical best and worst case scenarios to see how the network performs as machines are added and removed from the system.

4. Analysis and Results

This chapter presents the results of the simulation of the mechanism described in Chapter 3. Section 4.1 looks at analytical performance of a logging infrastructure that is set up with a single central server. It is configured to match the proposed topologies and is simulated with 10,000 machines on the network. Section 4.2 presents the results of a network simulated with 10,000 machines organized in to a tree of non-buffered peer groups. Section 4.3 presents the results of a network simulated with 10,000 machines organized in to a tree of buffered peer groups. Section 4.4 compares the performance of each of the topologies using the measurements defined in Chapter 3.

4.1. Central Server Performance

In order to have an established baseline to compare with, the theoretical performance of a network logging infrastructure based on sending all logs to a central server is presented below. The network is a basic setup where every machine sends log entries to the central server. A simple representation of this is presented in Figure 2.

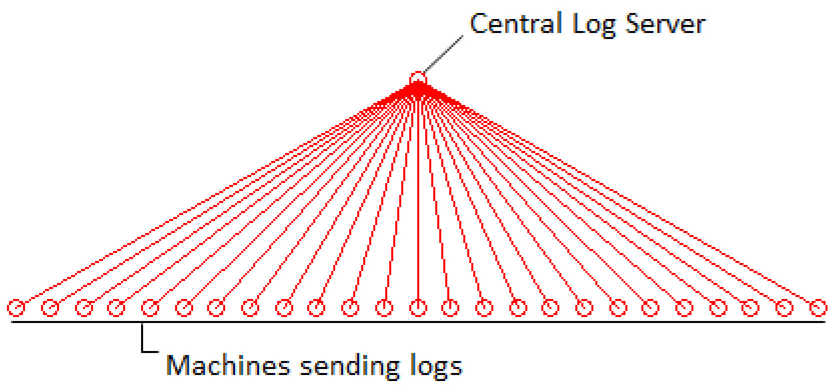

Figure 2 A sample logging infrastructure with a central log server and 25 client machines.

4.1.1. Search Complexity

The central server topology provides extremely efficient searching of the log files. All of the logs for the network are stored locally to the central server. From a network perspective, the query is a O(1) lookup off the local storage.

4.1.2. Redundancy

The central server topology has two different cases for message redundancy. In the best case, all log entries arrive at the log server and are recorded successfully. As long as the logs on the other machines in the network remain intact, two copies of each log entry will exist, one at the central server and one at the originating machine. However, since industry standard syslog operates over UDP with no confirmations, there is no guarantee that every log entry sent to the log server will arrive and get stored. At the client machine, settings can be configured to delete entries after a certain amount of time, when they reach a certain size, or to have a maximum size and overwrite the older entries. In addition, a threat to the network often works to compromise the system logs in order to hide the fact that he exists. These conditions make it so that it cannot be guaranteed that a log entry will still exist on the client machine when it is needed.

Due to the operating characteristics of the central server described above, a best case of two copies of a log entry and a worst case of zero copies of a log entry can exist on the network. It is expected that there will normally exist two copies of a message.

4.1.3. Event throughput and network usage

For the central server topology, there are two machine types to evaluate. The first machine type is a normal client machine on the network. Each machine on the network will generate a expected number of events per unit time R. This means that each machine on the network will have to send messages to the central server at a rate of R. The second machine type is the server receiving messages. This machine will be receiving messages at a rate or R for each of N machines on the network. This translates in to being able to receive and store messages at a rate of $R * N$.

4.1.4. Fairness

For the central server, there are the same two types of machines from above. There will be N machines that are sending messages at a rate of R and one server that receives messages at a rate of $R * N$. This shows that the number of messages that the server must handle increases linearly with N and is $O(N)$.

4.1.5. Summary

The analysis above gives us the characteristics for a central server based logging topology shown in Table 3.

Table 3 Summary of metrics for a central server based logging topology where N is the number of machines on the network and R is the rate at which they create and send log entries.

Searching the logs	O(1) local lookup
Log entry redundancy best case	2
Log entry redundancy worst case	0
Event processing best case	O(1), equal to R
Event processing worst case	O(N), equal to R*N
Fairness – Best case	N machines only sending messages
Fairness – Worst-case	1 machine receiving R*N messages
Network Traffic – Client machine	R messages being sent per unit time
Network Traffic – Server Machine	Receiving R*N messages per unit time

4.2. Tree of Non-Buffered Peer Groups

The first new mechanism to be analyzed is the tree of non-buffered peer groups described in Chapter 3.

4.2.1. Search Complexity

The search complexity of the peer groups is based on the size of the tree structure formed by the methodology described in Chapter 3. The measurements to quantify the search complexity of the tree are also described in Chapter 3. The mean number of peer groups in the network for the non-buffered topology is shown in

Figure **3**. The mean is centered on the x marks with a 95% confidence interval from 15 simulations hash marked above and below each x.

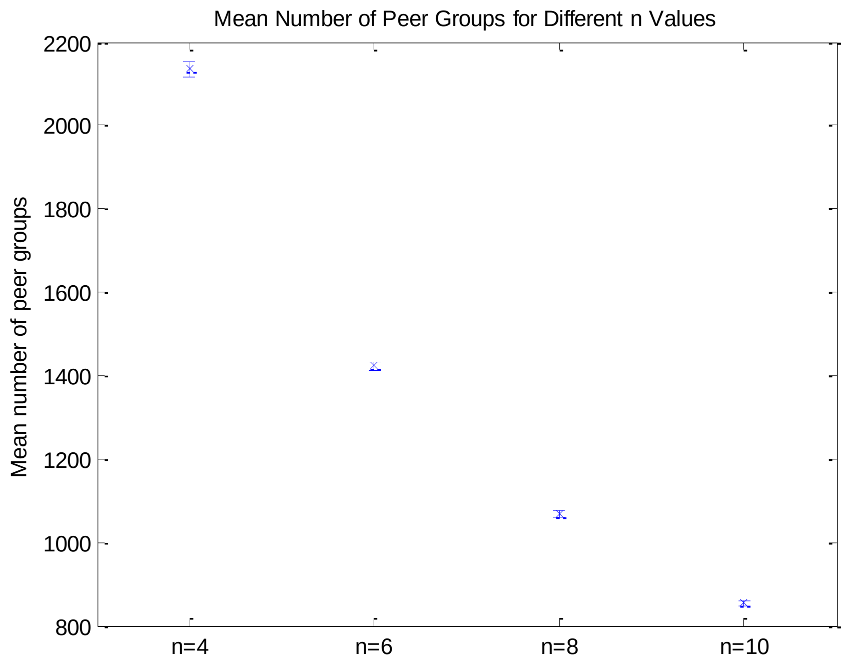

Figure 3: Mean Number of Peer Groups for Scenario 1.

4.2.2. Redundancy

The redundancy of the messages is directly related to the peer group size as described in Chapter 3. As the topology is designed, the non-buffered design ensures that every peer group is size n except for the root node. Since the root is designed and was observed to be the largest node in the tree. Every machine except members of the root had a redundancy of n.

4.2.3. Event throughput and network usage

Distributing all of the logs on the network to multiple places is expected to increase traffic on the network, and the traffic that most machines on the network will see. The more members a peer group has, the more traffic that the members will be having to process. The machines handling the most traffic will be the machines in the root peer group. Figure 4 plots the mean traffic expected to be processed by a machine in the root peer group, and displays a 95% confidence interval for the samples. The confidence intervals grow as n is increased because the non-buffered topology root node varies in size from n to $2 * n$ in size.

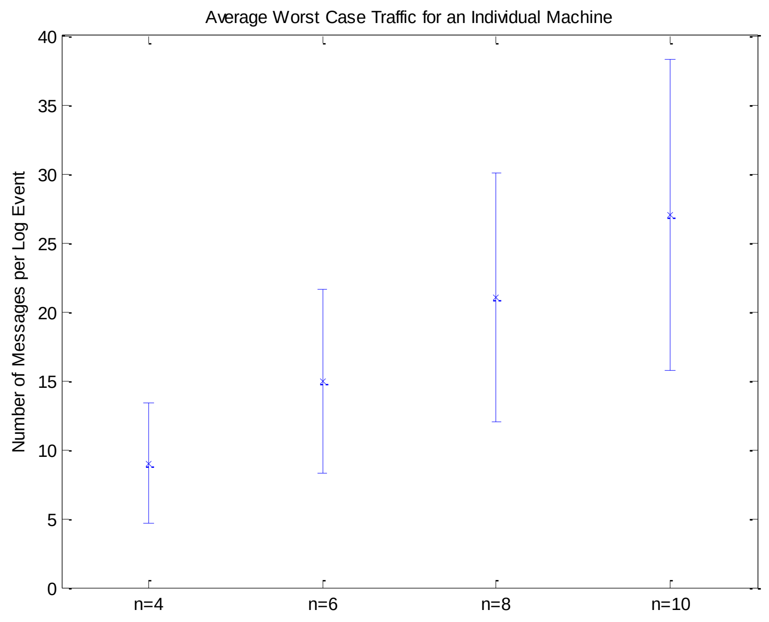

Figure 4: The maximum events per log event that any machine on the network will receive compared with a central server for the non-buffered topology.

The second measurement of traffic is the traffic on the entire network. The values shown in Figure 5 shown values are the average peer group size for each value of n along with the expected peer group traffic.

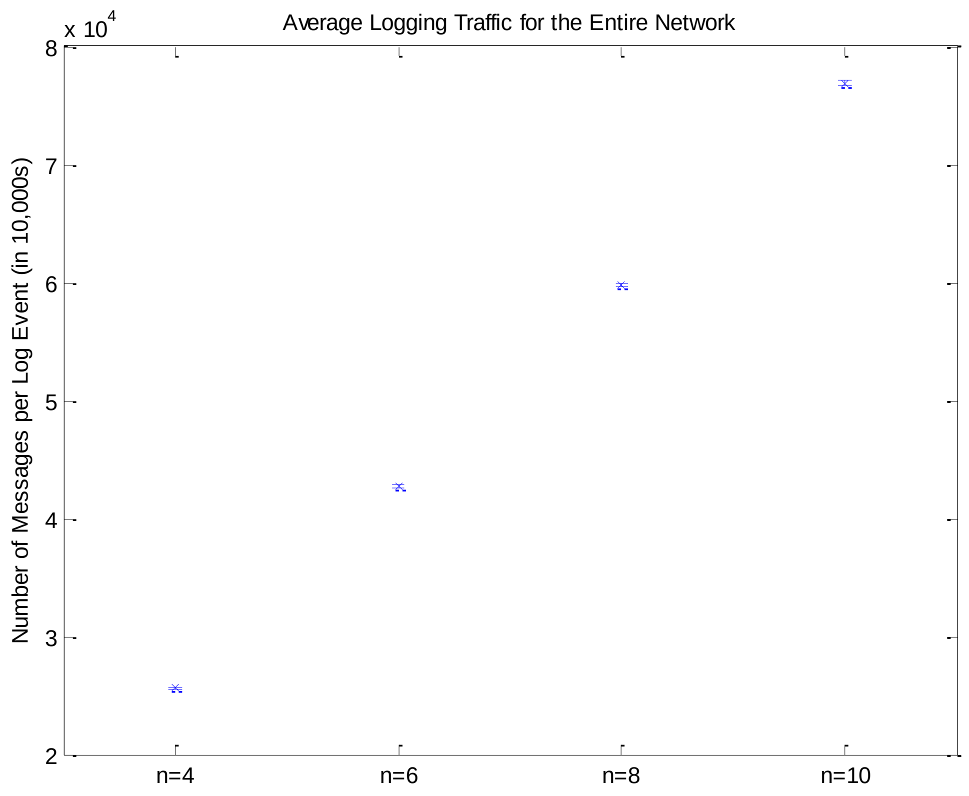

Figure 5: This graph compares the events per time that are on the network as a function of PG size for this topology.

4.2.4. Fairness

Fairness as described in Chapter 3 is measured by the difference in number of peer group changes between the machine with the most changes and the least changes. The data averaged over the 15 runs with a 95% confidence interval to predict the next simulation's fairness is shown in Figure 6.

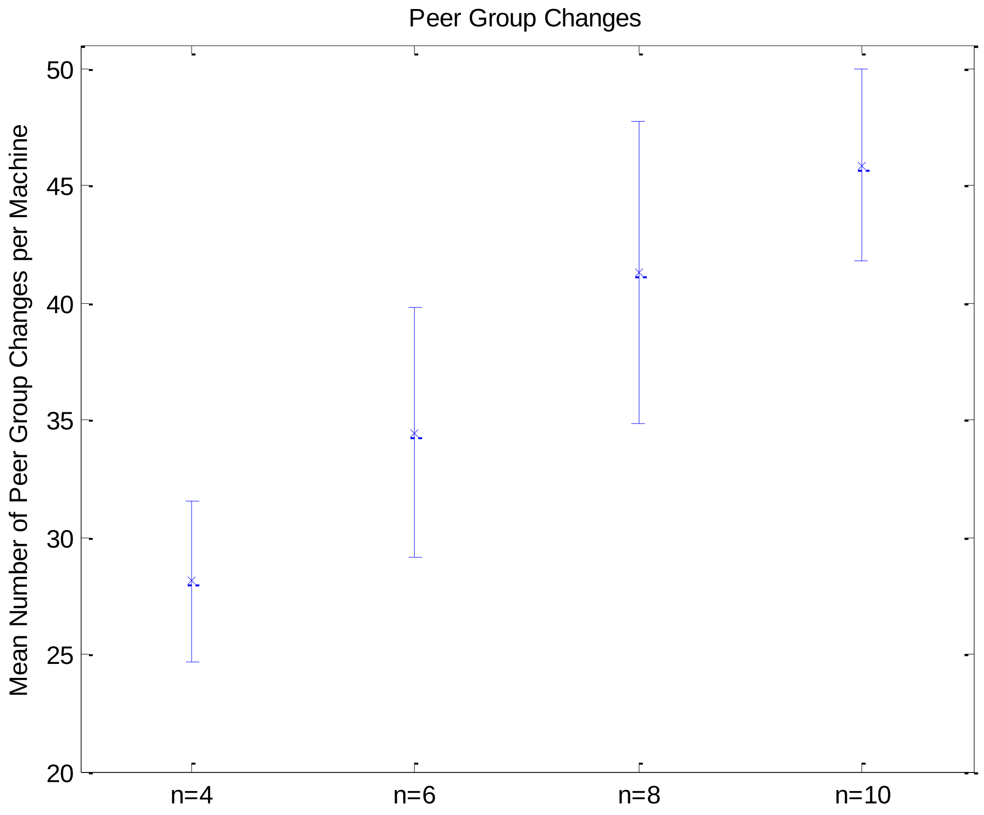

Figure 6: Average number of peer group changes per machine as a function of peer

.

4.3. Tree of Buffered Peer Groups

The second topology to be analyzed is the tree of buffered peer groups described in Chapter 3.

4.3.1. Search Complexity

The search complexity of the peer groups is based on the size of the tree structure formed by the methodology described in Chapter 3. The measurements to quantify the search complexity of the tree are also described in Chapter 3. The mean number of peer groups in the network for the buffered topology is shown in

Figure **3**. The mean is centered on the x marks with a 95% Confidence interval hash marked above and below each x.

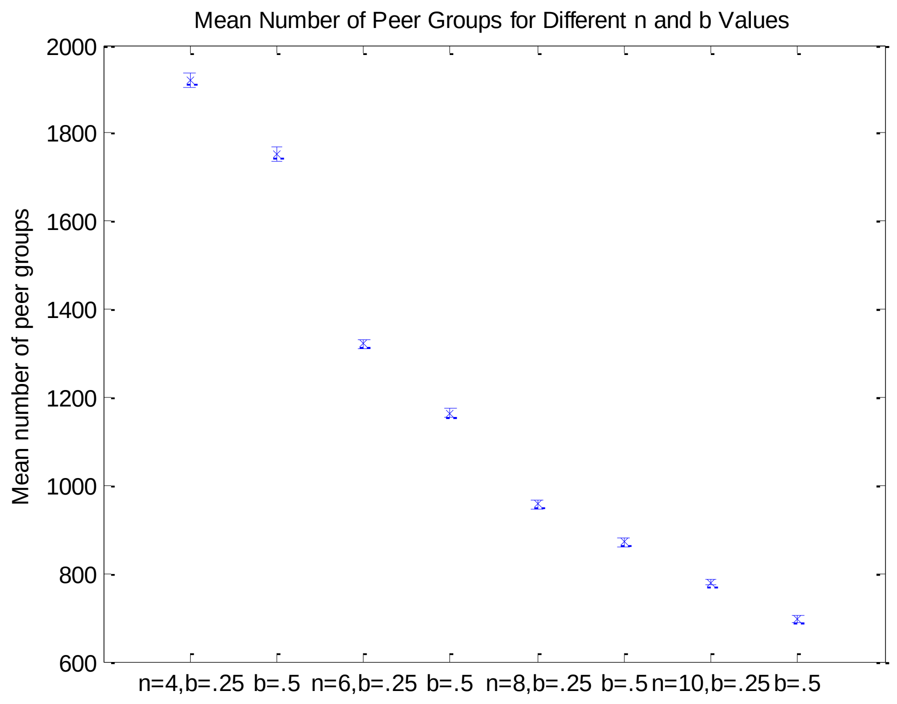

Figure 7: Mean Number of Peer Groups with buffered peer groups.

4.3.2. Redundancy

The redundancy of the messages is directly related to the peer group size as described in Chapter 3. A set of histograms for the proportion of peer group sizes for each configuration is shown in Figure 8. In the bottom two graphs, there were data points that were greater than sixteen, but all of those samples were for the root PG. Since there is only one root peer group, and higher is better, that data is left out of Figure 8.

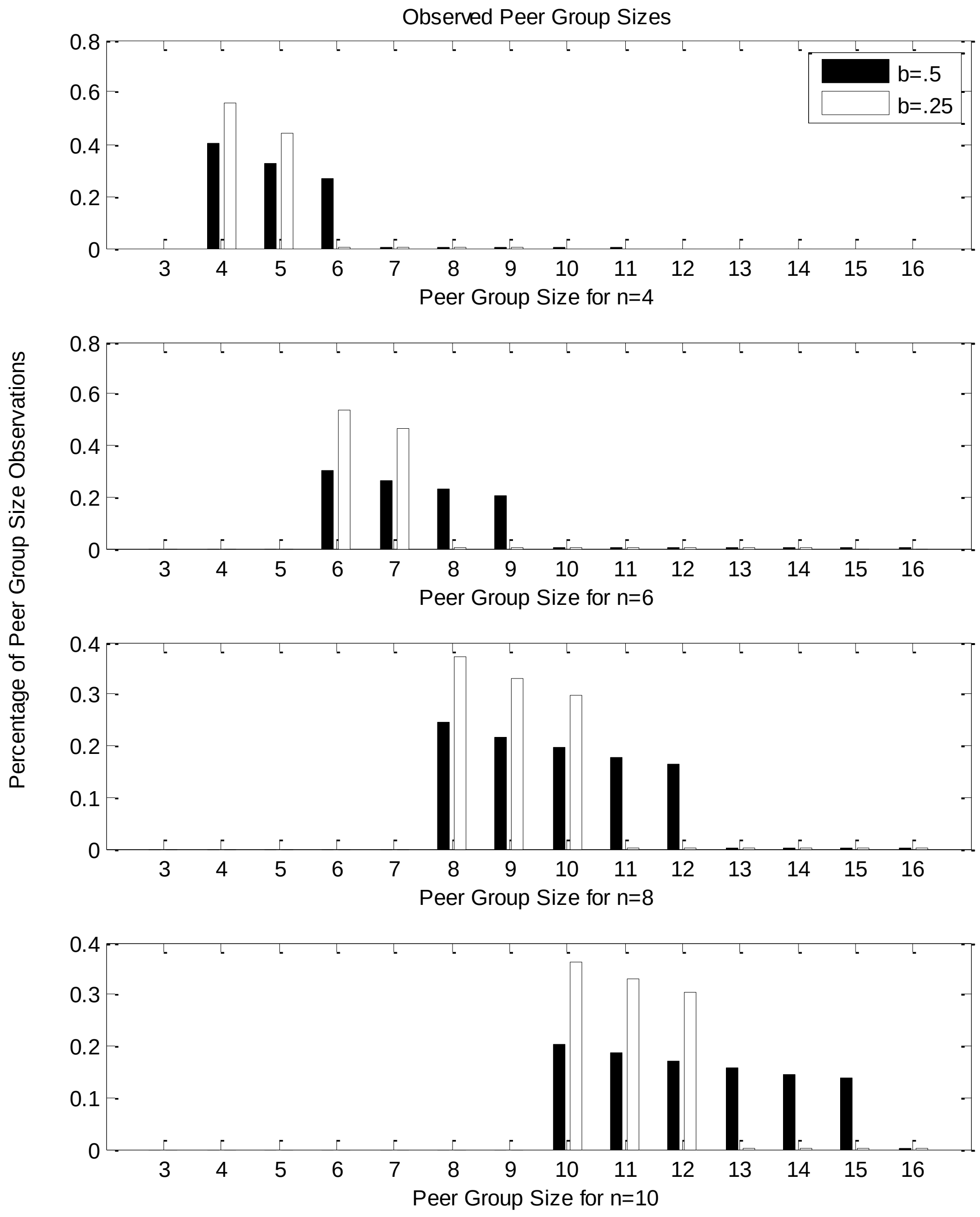

Figure 8: Average Peer Group Size for Scenario 1.

4.3.3. Event throughput and network usage

Distributing all of the logs on the network to multiple places is expected to increase traffic on the network, and the traffic that most machines on the network will see. The more members a peer group has, the more traffic that the members will be having to process. The machines handling the most traffic will be the machines in the root peer group. Figure 9 plots the mean traffic expected to be processed by a machine in the root peer group, and displays a 95% confidence interval for the samples.

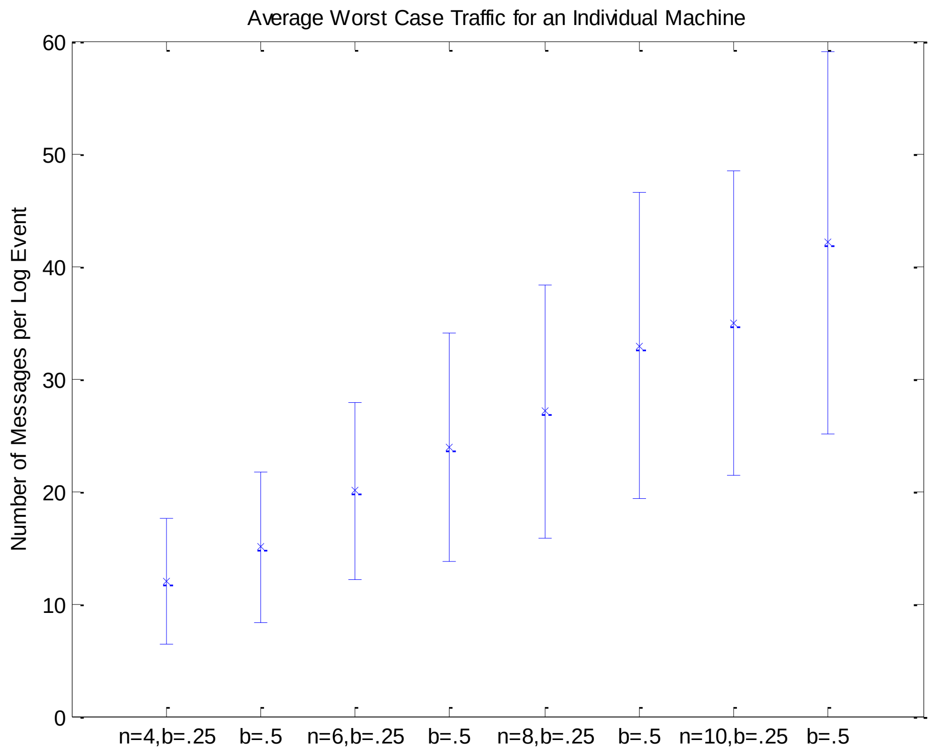

Figure 9:This graph compares the maximum events per log event that any machine on the network will receive compared with a central server for this topology.

The second measurement of traffic is the traffic on the entire network. The shown values are the average peer group size for each value of n and b along with the expected peer group traffic.

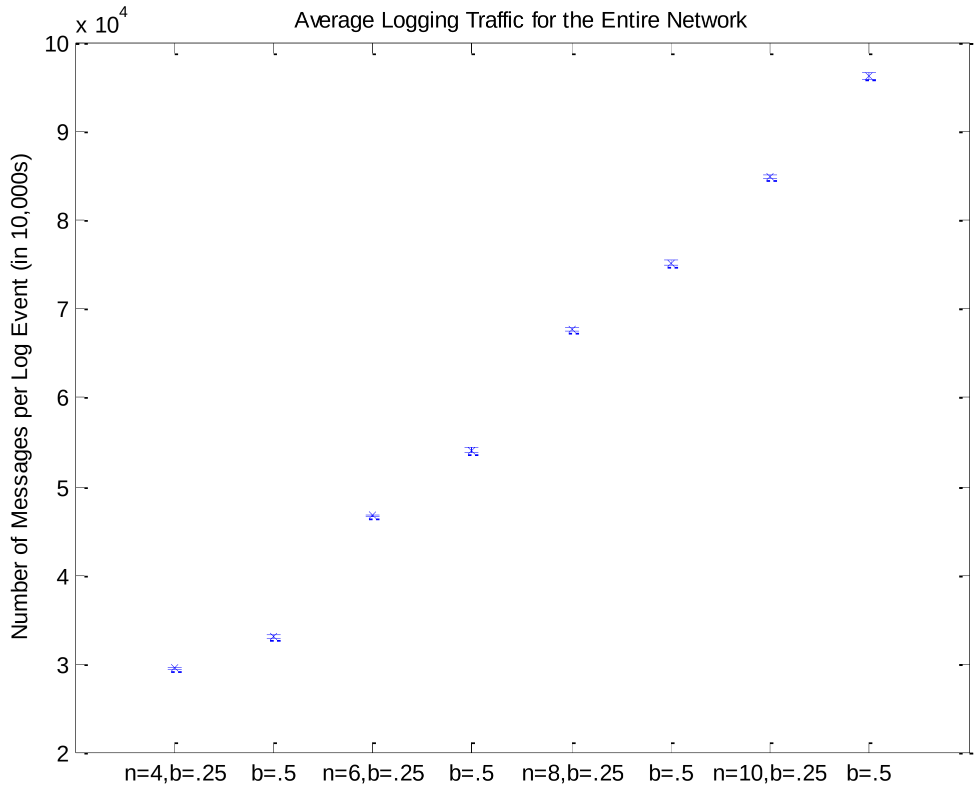

Figure 10: This graph compares the events per time that are on the network as a function of PG size for this topology.

4.3.4. Fairness

Fairness as described in Chapter 3 is measured by the difference in number of peer group changes between the machine with the most changes and the least changes. The data averaged over the 15 runs with a 95% confidence interval to predict the next simulation's fairness is shown in Figure 11.

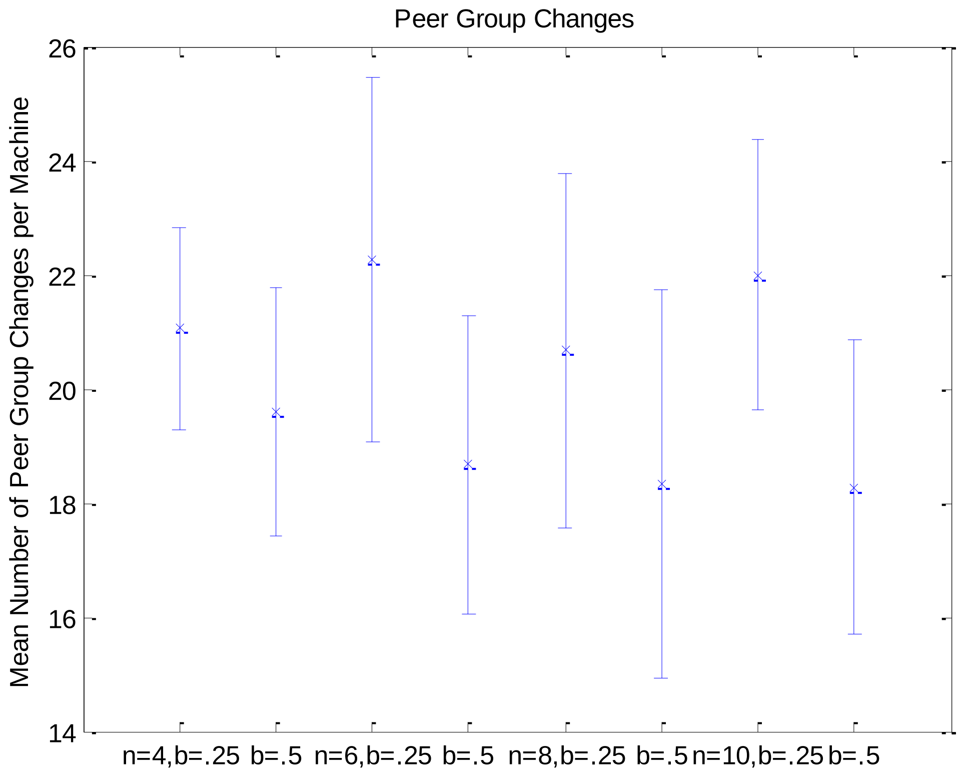

Figure 11: Averaged number of peer group changes that machines had to make with various *n* and *b* values.

4.4. Performance Analysis

4.4.1. Search Complexity

For comparing the search complexity of the different topologies, the central server is the most efficient, and provides the best performance. This is expected because all of the logs being searched are on one machine. For the distributed topologies, the buffered groups were better for searching because there were fewer peer groups when comparing topologies with the same n parameter.

4.4.2. Redundancy

The redundancy of the logs was found to be best in the buffered peer group topology. The reason for this topology being the best is that it ends up with slightly larger peer groups because of the buffer which results in extra copies of the logs being on the network. The non-buffered topology performed second best and the central server was the worst.

4.4.3. Network Traffic

For the worst case traffic that any individual machine must handle, the non-buffered topology had the most traffic since it had the smallest maximum peer group size in the root. The buffered peer group topology was close behind it having an only slightly larger root peer group. The central server topology is the worst for this metric because one machine must receive traffic from every machine on the network.

The overall traffic on the network was found to be larger on both of the distributed topologies; however this was expected due to the nature of distributing the logs to multiple places. The overall traffic on the network is still increasing proportionally to the size of the network, and this should not be a problem on a standard wired network.

4.4.4 Fairness

The fairness of the system found that the buffered peer groups performed better than the non-buffered peer groups. The non-buffered topology had to move all members of a peer group every time a single member goes offline. The buffer provided several extra machines that had to go offline before the rest of the peer group had to be moved, and significantly improved the network performance.

5. Conclusions and Recommendations

This chapter summarizes the findings and results presented in Chapter 4. The results are used as a basis to determine future areas that need to be researched. The goal of this research was to propose a solution to providing an efficient way to have network logs distributed between machines generating logs instead of sending them all to a central repository. A system was proposed in Chapter 3 to meet this goal, and portions of it tested in Chapter 4.

5.1. Conclusion and Findings

The simulations showed that the proposed system has several very desirable qualities. The system establishes groups of machines that synchronize network traffic and set up a system where the amount of traffic that any individual machine on the network must process is independent of the size of the network. In the process of setting up this log distribution, the amount of traffic across the whole network is greater than that in the central server model, but it still grows at a rate proportional to the size of the network (linear growth).

Two different topologies were tested. The buffered peer group topology showed an improvement in all desired characteristics except for total network traffic volume. The increase in traffic was seen to be insignificant compared to the improvements in other measurements.

5.2. Future Work

Some of the areas of this research that need further work, are described below. For this topology, the amount of traffic within a peer group is a quadratic function of the peer group size. This would generally not be significant when machines are on the same physical network segment, but research to test what happens when machines are on different physical segments should be done. It may be necessary to ensure that members of a peer group are on the same network segment or subnet to prevent quadratic growth of traffic throughout the entire network.

The simulations performed for this research were not able to explore performance of the system on actual hardware. Research to determine how long a query takes and how often queries can be performed needs to be measured on higher fidelity models (such as NS2 or OPNET) or possibly on actual hardware.

Research for streaming of "live" content via distributed topologies such as BitTorrent would apply to this problem.

The number of copies of a log entry that would be stored on the network when compared to a central server model (depending on what the n parameter is configured at) should be tested and compared. In the event of a discrepancy between the copies of the logs, the originator would be notified of the discrepancy which is a service not performed by a central server.

Bibliography

Androutsellis-Theotokis, S., & Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Comput.Surv.*, *36*(4), 335-371. Retrieved from http://doi.acm.org/10.1145/1041680.1041681

Cohen, B. (2003). Incentives build robustness in BitTorrent. *1st Workshop on Economics of Peer-to-Peer Systems.*

Denning, D. E. (1999). *Information warfare and security* Addison-Wesley.

Federal Information Security Management Act, (2002).

Frankel, J., & Pepper, T. *Gnutella protocol specification.*http://wiki.limewire.org/index.php?title=GDF

Fu, Q., Lou, J., Wang, Y., & Li, J. (2009). Execution anomaly detection in distributed systems through unstructured log analysis. Paper presented at the *ICDM '09: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining,* 149-158. Retrieved from http://dx.doi.org/10.1109/ICDM.2009.60

GadAllah, S. (2004). The importance of logging and traffic monitoring for information security.

Gordon, L. A., Loeb, M. P., Lucyshyn, W., & Richardson, R. (2006). *2006 CSI/FBI computer crime and security survey*Computer Security Institute.

Greitzer, F. L., Moore, A. P., Cappelli, D. M., Andrews, D. H., Carroll, L. A., & Hull, T. D. (2008). Combating the insider cyber threat. *IEEE Security and Privacy*, *6*(1), 61-64. Retrieved from http://doi.ieeecomputersociety.org/10.1109/MSP.2008.8

Halonen, P., Miettinen, M., & Hatonen, K. (2009). Computer log anomaly detection using frequent episodes. *Proceedings of the 5TH IFIP Conference on Artificial Intelligence Applications and Innovation*. 417-422.

Hayden, M. V. (1999). The insider threat to U. S. government information systems. *Report from NSTISSAM INFOSEC /1-99,*

Holz, T., Gorecki, C., Rieck, K., & Freiling, F. C. (2008). Measuring and Detecting Fast-flux Service Networks. *Proceedings of the Network and Distributed System Security Symposium.*

Jiang, H., Li, J., Li, Z., & Bai, X. (2008). Performance evaluation of content distribution in hybrid CDN-P2P network. Paper presented at the *FGCN '08: Proceedings of the 2008 Second International Conference on Future Generation Communication and Networking*, 188-193. Retrieved from http://dx.doi.org/10.1109/FGCN.2008.63

Karrels, D., Peterson, G., & Mullins, B. (2009). Structured P2P technologies for distributed command and control. *Peer-to-Peer Networking and Applications,* Retrieved from http://dx.doi.org/10.1007/s12083-009-0033-y.

Krishnamurthy, B., Wills, C., & Zhang, Y. (2001). On the use and performance of content distribution networks. Paper presented at the *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, San Francisco, California, USA. 169-182. Retrieved from http://doi.acm.org/10.1145/505202.505224

Leibowitz, N., Ripeanu, M., & Wierzbicki, A. (2003). Deconstructing the Kazaa Network. *Proceedings of the Third IEEE Workshop on Internet Applications.*

Levoy, T. E. (2006). Development of a methodology for customizing insider threat auditing on a microsoft windows XP operating system. Air Force Institute of Technology).

Li, J. (2008). On peer-to-peer (P2P) content delivery. *Peer-to-Peer Networking and Applications, 1*(1), 45-63. Retrieved from 10.1007/s12083-007-0003-1; http://dx.doi.org/10.1007/s12083-007-0003-1

Ma, D., & Tsudik, G. (2009). A new approach to secure logging. *ACM Transactions on Storage, 5*(1), 1-21. Retrieved from http://doi.acm.org/10.1145/1502777.1502779

Mills, R. F., Peterson, G. L., & Grimaila, M. R. (2009, March). Insider threat prevention, detection, and mitigation. *Cyber-Security and Global Information Assurance: Threat Analysis and Response Solutions.*

Monteiro, S. D. S., & Erbacher, R. F. (2008). An authentication and validation mechanism for analyzing syslogs forensically. *SIGOPS Oper.Syst.Rev., 42*(3), 41-50. Retrieved from http://doi.acm.org/10.1145/1368506.1368513

Pallis, G., & Vakali, A. (2006). Insight and perspectives for content delivery networks. *Commun.ACM, 49*(1), 101-106. Retrieved from http://doi.acm.org/10.1145/1107458.1107462

Peisert, S., Bishop, M., & Marzullo, K. (2008). Computer forensics in forensis. *SIGOPS Oper.Syst.Rev., 42*(3), 112-122. Retrieved from http://doi.acm.org/10.1145/1368506.1368521

Richardson, R. (2008). *2008 CSI/FBI computer crime and security survey*Computer Security Institute.

Schneier, B., & Kelsey, J. (1999). Secure audit logs to support computer forensics. *ACM Trans.Inf.Syst.Secur., 2*(2), 159-176. Retrieved from http://doi.acm.org/10.1145/317087.317089

Shenk, J. (2008). Demanding more from log management systems, from http://www.sans.org/reading_room/analysts_program/LogMgt_June08.pdf

SolarWindows, I. (2009). *INFO: How many messages can kiwi syslog server handle?* Retrieved Oct 14, 2009, from http://www.kiwisyslog.com/kb/info:-how-many-messages-can-kiwi-syslog-server-handle?/

Spafford, E. H. (1989). *The internet worm incident* Retrieved from https://www.cerias.purdue.edu/techreports-ssl/public/933.pdf

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 25-03-2010	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* March 2009 – March 2010
4. TITLE AND SUBTITLE A Distributed Network Logging Topology		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Fritts, Nicholas E., Second Lieutenant, USAF		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865		**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCO/ENG/10-07
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Network logging is used to monitor computer systems for potential problems and threats by network administrators. Research has found that the more logging enabled, the more potential threats can be detected in the logs. However, generally it is considered too costly to dedicate the manpower required to analyze the amount of logging data that it is possible to generate. Current research is working on different correlation and parsing techniques to help filter the data, but these methods function by having all of the data dumped in to a central repository. Central repositories are limited in the amount of data they are able to receive without losing some of the data. In large networks, the data limit is a problem, and industry standard syslog protocols could potentially lose data without being aware of the loss, potentially handicapping network administrators in their ability to analyze network problems and discover security risks. This research provides a scalable, accessible and fault-tolerant logging infrastructure that resolves the centralized server bottleneck and data loss problem while still maintaining a searchable and efficient storage system.

15. SUBJECT TERMS

Syslog, Network, Distributed Topologies

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Borghetti, Lt Col, USAF
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	69	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4612 (brett.borghetti@AFIT.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18